Compiling relational queries for a finite state automation hardware filter

D. Plateau

▶ **To cite this version:**

D. Plateau. Compiling relational queries for a finite state automation hardware filter. RR-0171, INRIA. 1982. inria-00076387

HAL Id: inria-00076387
https://inria.hal.science/inria-00076387

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

IRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tel 954 90 20

Rapports de Recherche

N° 171

COMPILING RELATIONAL QUERIES FOR A FINITE STATE AUTOMATON HARDWARE FILTER

Didier PLATEAU

Novembre 1982

COMPILING RELATIONAL QUERIES

FOR A FINITE STATE AUTOMATON HARDWARE FILTER

Didier PLATEAU

Abstract

Most data base machines use a hardware filter that performs the projection and selection operation on the fly during disk transfer. Many possible designs have been proposed for such filters and a promising approach seems to be to use a finite state automaton. In this paper we briefly describe such a filter and we present an algorithm that compiles selection/projection queries into finite state automata (FSA). A major drawback of the approach is the size of the FSA, therefore we give an evaluation of the number of states of the automaton.

Résumé

La plupart des machines base de données utilisent un filtre câblé capable de réaliser les opérations de sélection et de projection "au vol", durant le transfert des données du disque vers la mémoire centrale. Diverses solutions peuvent être adoptées pour réaliser ce filtre. L'utilisation d'un filtre de type automate à états finis nous semble intéressante. Dans ce papier, nous décrivons brièvement un tel filtre puis nous présentons un algorithme qui compile des requêtes de sélection projection en un automate à états finis. L'inconvénient majeur de cette approche est la taille de l'automate. Nous proposons une évaluation du nombre d'états de l'automate.

PAPIER RECUPERE ET RECYCLE

I. INTRODUCTION

Most data base machines (DBM) use some kind of filtering mechanism to perform unary operations on relations. These filters are assumed to process data on the fly during its transfer from disk into main memory. Among the many suggestions for the design of such filters the finite state automaton approach seems to be most promising. J. ROHMER [2] justifies the idea of a FSA filter for a DBM and gives examples of practical application. P. FAUDEMAY [3] suggests a new class of filters derived from the FSA approach. In [4] the functional specifications of a filter that is used in the VERSO DBM was presented. Other implementations of filters were suggested in [8] by Haskin.

Some actual implementations of filters were also undertaken.

The CAFS machine [5] (content Addressable File Store) is a special purpose peripheral designed principally to handle real-time transactions in a multiuser environment.

The SARI machine [6] (Systeme Associatif de Recherche d'Information) was implemented to allow complex requests with a large interval data flow, especially for a telephone information application.

The TREFLE machine [7] is a special-purpose Information Retrieval machine, dealing with data structure which can be described by algebric grammar.

The SURE processor at the University of Brunshweig [9] is based on parallel processing of information flow.

The main advantages of FSA filtering are :

(1) The processing time of an FSA filter is linear in the size of the processed data i.e. $T(n) = Kn$, this allows on the fly filtering.

(2) The constant K is independent from the automaton used, i.e. this allows on the fly filtering of any query independent of its complexity.

(3) Finite State Automata are well known object and we can rely on automata theory to solve related problems (existence, minimization, etc.)

Query execution in a hardware filter environment consists in three phases

(1) Compiling (i.e generating the FSA from the query)

(2) Loading (i.e loading the FSA description into the filter)

(3) Processing (i.e actual filtering of the data).

There is a general concensus to recognize that processing time is very good : in fact it cannot be better because it is done at disk transfer rate and any speed increase would mean that the filter would wait for the disk. On the contrary compiling time might be a problem if compiling happens to be a complex task and loading time might be also a bottle neck if automaton size is too big.

This paper deals with these two problems to show the feasibility of the FSA approach

(1) We describe an actual implementation of a compiler

(2) We give evaluations of the automaton size to show that for "reasonable queries" the FSA size is acceptable.

II. PROBLEM DESCRIPTION

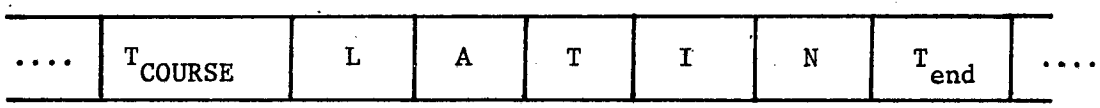
II.1. Data Representation

We shall assume the reader familiar with relational terminology. Data consists of relations specified by descriptions of the form $R(A_1,A_2,\ldots,A_n)$. The physical representation of attributes is as follows :

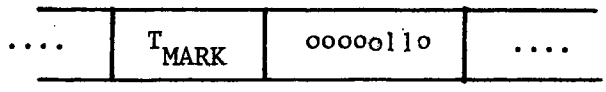
An attribute value can have a fixed or variable size : In the first case, a special character (or byte) marks the beginning of the attribute. This byte characterizes the attribute and its format. In the second case, there is also a tag marking the end of the value.

Example :

1) The attribute COURSE is a character string with variable length.
   The value LATIN will be represented by :



2) the attribute GRADE had a binary representation (Its length must be fixed). The value 12 has the representation



Notice that the only character which is forbidden in a variable length attribute value is the end character. So that we are not limited by the alphabet size.

In the sequel, we shall omit these characters whenever possible.

The physical representation of a relation is a sequence of records. The structure of a record is characterized by a regular expression called the relation format.

For instance, if we consider the following relation :

R	COURSE	STUDENT	GRADE
	Mathematics	Toto	12
	Mathematics	Toto	15
	Mathematics	Lulu	13
	Mathematics	Lulu	18

it can have the following formats :

(1) (COURSE,STUDENT,GRADE)* meaning that each record is a tuple. The representation of R is :

MATHEMATICS TOTO 12 MATHEMATICS TOTO 15
MATHEMATICS LULU 13 MATHEMATICS LULU 18

(2) (COURSE(STUDENT(GRADE)*)*)* meaning that each record consists of :

- a COURSE ;
- the list of the STUDENTS who are attending to this COURSE,
- for each STUDENT, the list of the GRADE he has got for this COURSE.

The representation of R is then :

MATHEMATICS TOTO 12 15 LULU 13 18

II.2. Operations

The operations we want to perform are all the unary operations of the relational algebra. But, in this paper, we restrict ourself to the selection of the tuples which satisfy a boolean condition on the attributes and the projection on some of the attributes of the relation. This condition has an arbitrary complexity.

Example : Let R(COURSE,STUDENT,GRADE) denote a relation. An example of operation is :

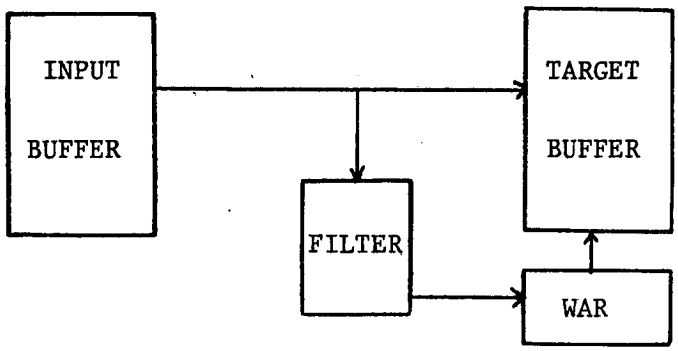
Get the COURSE such that STUDENT = TOTO and GRADE > 12 or STUDENT = LULU and GRADE > 16.

II.3. Filter Description

The filter processes the data during its transfer from an input buffer (that could be the disk) to a target buffer.

The filter controls the writing address register (WAR) of the target buffer.

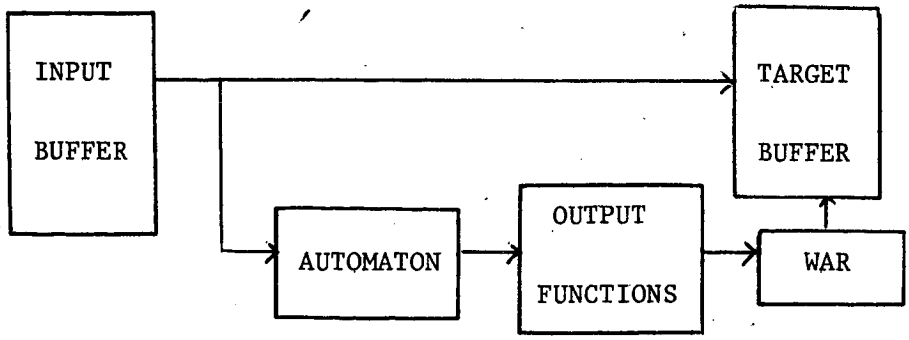
Figure 1. Logical schema of the VERSO filter



The filter consists of

- the automaton defined by its transition matrix
- a set of output functions which control the write operation in the target buffer with the WAR.

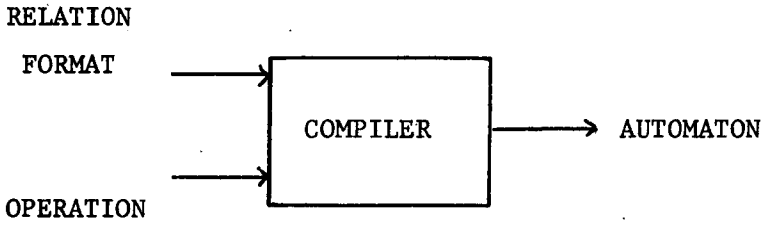
Figure.2.



At time t, the automaton, in state Q_t, reading the input character C_t, activates the list of output functions S_t and loads the next state Q_{t+1}. For a more complete description of the filter, see [1].

III. COMPILER DESCRIPTION

Given a relation format, and a selection-projection operation, the compiler has to generate the automaton which computes the resulting relations.



In the generation phase of the automaton, the compiler has to solve two problems :

(1) substrings recognition
(2) condition evaluation

Example : Let R(COURSE,STUDENT,GRADE) denote a relation with the format $(COURSE(STUDENT(GRADE)^*)^*)^*$ and consider the operation :

Get the COURSE such that

STUDENT = TOTO and GRADE $\geq$ 12 or
STUDENT = LULU and GRADE > 16

Let R have the value :

MATHEMATICS TOTO 12 15 LATIN MAX 11

The automaton has

(1) to recognize the substrings TOTO,LULU,12,16
(2) to decide that MATHEMATICS TOTO 15 is the only tuple which satisfies the selection condition and to remember when the GRADE value 15 is read that the corresponding tuple is MATHEMATICS TOTO 15.

(this means that a state memorizes the tuple being read).

In our implementation, the compiler will in a first step, generate the two subautomata which recognize TOTO and LULU on one hand, 12 and 16 on the other hand.

The second step consists in the generation of the automaton which uses these two subautomata to evaluate the selection condition.

Substrings recognition

At the attribute level, the automaton has to recognize the position of the filtered strings in comparison with the parameters strings of the selection condition.

Formally let A denote an attribute of a relation R and

$$(A\ c_i\ \mu_i)_{1 \le i \le n}$$

be the elementary conditions on A such that

$$c_i \in \{=,>,<,\geq,\leq,\neq\}$$

and $\mu_i \in \Sigma^*$ where Σ is an alphabet. We assume that Σ^* is lexicographically ordered and that

$$\mu_i < \mu_{i+1} \quad 1 \le i \le n-1$$

We define $I_1 = \{\mu \in \Sigma^* \ / \ \mu < \mu_1\}$

$$I_{2i} = \{\mu_i\} \quad 1 \le i \le n$$

$$I_{2i+1} =]\mu_i, \mu_{i+1}[\quad 1 \le i \le n-1$$

$$I_{2n+1} = \{\mu \in \Sigma^* \ / \ \mu > \mu_n\}$$

Then $\{I_j\}_{1 \le j \le 2n+1}$ is a partition of Σ^*

We define $f : \Sigma^* \rightarrow \{0,1\}^n$

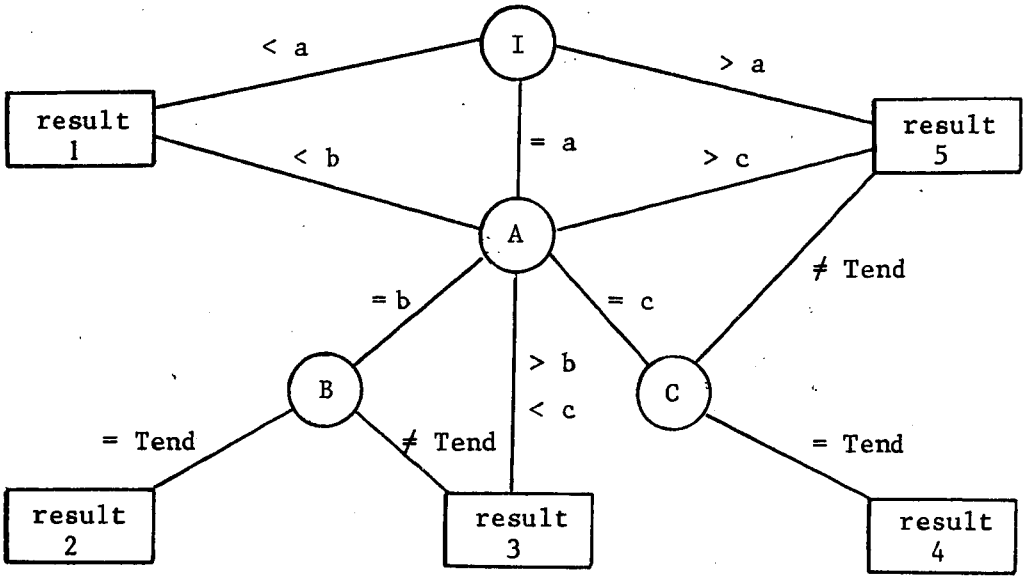
$\mu \rightarrow (b_1,-,b_n)$ such that

$b_i = (\mu\ c_i\ \mu_i)$

Then the function f keeps the same value in each I_j.
So the knowledge of which I_j the filtered strings belongs to is sufficient to set a boolean value to each elementary condition.

The subautomaton which filters the A value, will compute the number j such that the filtered string belongs to I_j.

Example : $\mu_1 = ab$; $\mu_2 = ac$
The subautomaton will be



We can notice that the subautomaton looks like the lexical tree which represents the strings $(\mu_i)_{1 \leq i \leq n}$

The subautomaton will be a subroutine of the whole automaton. The reason of this choice is space minimization.

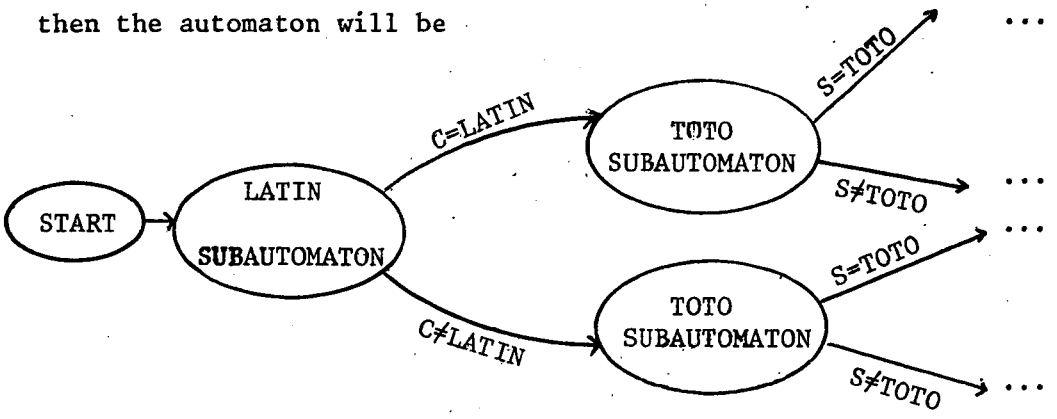
Example : Let R(COURSE,STUDENT,GRADE) be a relation with a format

$$(COURSE(STUDENT(GRADE)^*)^*)^*$$

and an operation

Get the tuples such that
COURSE = LATIN and GRADE = 15 or
STUDENT = TOTO

then the automaton will be



We can see that the subautomaton which recognizes TOTO is used twice. The subroutine mechanism allows us to load it just one time, thus saving space.

This mechanism implies that we are able to force the internal states at the end of a subroutine and to force the input character if the subroutine is a function.

Compared with figure 2 we have a new logical schema (see figure 3).

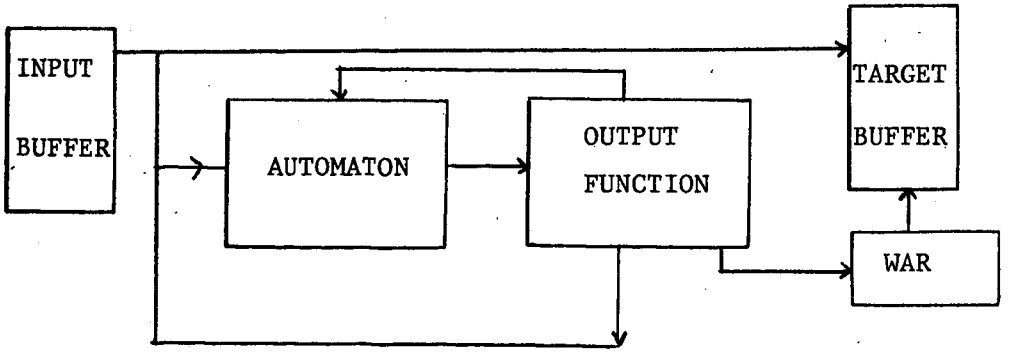


Figure 3.

Evaluation of the condition

The second phase of the compilation is the generation of that use the subroutines to evaluate the selection condition. These states are called logical states.

The role of logical states is :

- to call the subroutine
- to interpret the result of the subroutine
- to memorize the evaluation of the boolean expression

The main problem is the third point. The semantics of a logical state must be sufficient to evaluate the condition for each tuple and to start the evaluation in the middle of a tuple because of the record structure.

Semantics of a logical state

Let $C = T_1$ or ... or T_p be the boolean expression, Q the set of the logical state of the automata.

We define $C : Q \rightarrow \{0,1\}^p$

$q \rightarrow (t_1,-,t_p)$ such that

$\forall\, i \in \{1,..,p\}$ $t_i = 0$ iff in state q T_i is wrong

$t_i = 1$ otherwise

The predecessor of state q is state q' such that the transition $(q' \rightarrow q)$ belongs to the automaton.
We denote $q' = pred(q)$.

Then, the knowledge of the two functions C and q is sufficient to solve our problem.
At the end of a tuple t, if we are in state q we can decide if t satisfies the condition with the algorithm :

Let $c(q) = (t_1,\ldots,t_p)$

if $\forall\, i \in \{1,-,p\}\ t_i = 0$ then t doesn't satisfy the condition

else t satisfies the condition.

On the other hand, we can start a new evaluation in the middle of a tuple by following the list of the q predecessors.

Generation of the logical states

The generation of the logical states goes through several steps.

Let $R(A_1,-,A_N)$ be a relation.

Let q_o be the initial state of the automaton.

Let LS_i be the list of the logical states, representing all the possible evaluations after the A_i recognition.

Let $R_i = 1,\ldots,c_i$ the set of the possible results of the subroutine which filters attribute A_i.

The generation algorithm operates as follows :

Step 0 $c(q_o) \leftarrow (1,\ldots,1)$;
$pred(q_o) \leftarrow 0$;
$LS_o \leftarrow \{q_o\}$
$i \leftarrow 1$

Step 1 $\forall\, q \in LS_{i-1}$, $\forall\, r \in R_{io}$

* compute the semantics of the logical state q' issued from q and r

* if such a state is not yet in LSi then

LSi ← LSi ∪ {q'}

If i < N then i ← i+1 go to step 1.

On the other hand, optimization strategies are implemented (success or failure before the end of the tuple) to optimize the automaton size.

IV. AUTOMATON SIZE

As described in section 2, the automaton consists of the subroutines and the logical states. We first evaluate the subroutine size.

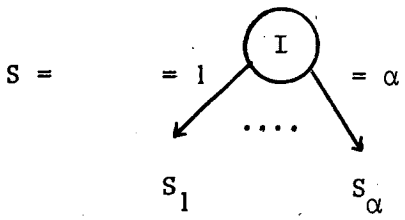
Evaluation of the subroutine size

Let A be an attribute of R and $(A\ c_i\ \mu_i)_{1 \le i \le N}$ be the elementary conditions on A where $c_i \in \{=,>,<, \neq, \neq, \neq\}$ and $\mu_i \in \Sigma^*$.

Let $|\Sigma| = \alpha$ be the alphabet size.

Let S be the subroutine which recognize the $(\mu_i)_{1 \le i \le n}$

Let S_i be the part of S which corresponds to the case where the first character read is an "i".



For our evaluation, we assume that $\mu_i \in \Sigma^{\ell}$ (where ℓ is fixed). We shall see that such an hypothesis is not restrictive.

Let $s(n,\ell)$ denote the subroutine size, $m(n,\ell)$ the minimum size, $M(u,\ell)$ the maximum size and $J(n,\ell)$ the average size of the subroutine.

The maximum value is reached when all the strings μ_i have different beginning characters. (This implies $n \le \alpha$).

Then

$$M(n,\ell) = 1 + n * \ell$$

The minimum value is reached when all the strings μ_i have the same $(\ell-1)$ first characters and a different last character. (We assume that all the strings μ_i are different).

Then $m(n,\ell) = n+\ell$;

To get an average value of $S(n,\ell)$, we have to make some probabilistic assumptions. We have chosen an uniform distribution model.

Formally, let $p(i,j,k)$ denote the probability that an "i" is the j^{th} character of string μ_k.
Then

$$\forall\, i \in \{1,2,\ldots,\alpha\}\ ,\ \forall\, j \in \{1,2,\ldots,\ell\},\quad \forall\, k \in \{1,\ldots,n\}$$

$$p(i,j,k) = \frac{1}{\alpha}$$

With these assumptions, we use a generating function method to compute the average size.
Let $\mathcal{S}_n^{(\ell)}$ denote the set of subroutines which recognize n strings belonging to Σ^ℓ

$$\mathcal{S}^{(\ell)} = \bigcup_{n=o}^{\alpha^\ell} \mathcal{S}_n^{(\ell)}$$

$\forall\, S \in \mathcal{S}^{(\ell)}$ node (S) = number of states of subroutine S.

$$\alpha_n^{(\ell)} = \sum_{S \in \mathcal{S}_n^{(\ell)}} \text{node (S)}$$

$$\text{NODE}^{(\ell)}(z) = \sum_{n=0}^{n=\alpha^\ell} \alpha_n^{(\ell)}\, z^n$$

$$\mathcal{D}^{(\ell)}(z) = \sum_{n=0}^{n=\alpha^\ell} |\mathcal{S}_n^{(\ell)}|\, z^n$$

$\forall\, S \in \mathcal{S}^{(\ell)}$ $|S|$ = the number of strings that subroutine S recognizes

Then we have

$$|\mathcal{S}_n^{(\ell)}| = \binom{\alpha^\ell}{n}$$

so that $\mathcal{D}^{(\ell)}(z) = (1+z)^{\alpha^\ell}$ (1)

$$\text{NODE}^{(\ell)}(z) = \sum_{n=0}^{n=\alpha^{\ell}} \alpha_n^{(\ell)} z^n$$

$$= \sum_{S \in \mathcal{S}^{(\ell)}} \text{node}(S)\, z^{|S|}$$

$$\text{But node}(S) = \sum_{i=1}^{i=\alpha} \text{node}(S_i)+1 \quad \text{if } S \neq \emptyset$$

$$\text{node}(\emptyset) = 0$$

$$\text{So NODE}^{(\ell)}(z) = \sum_{S \in \mathcal{S}^{(\ell)}} z^{|S|} - 1$$

$$+ \sum_{S \in \mathcal{S}^{(\ell)}} \left(\sum_{i=1}^{i=\alpha} \text{node}(S_i) \right) z^{|S|}$$

$$\text{But } |S| = \sum_{i=1}^{\alpha} S_i$$

Using the uniformity assumptions, we have :

$$\text{NODE}^{(\ell)}(z) = \mathcal{P}^{(\ell)}(z) - 1 + \sum_{S_1,\ldots,S_\alpha \in \mathcal{S}^{(\ell-1)}} \text{node}(S_i)\, z^{\sum_{i=1}^{\alpha} |S_i|}$$

$$= \mathcal{P}^{(\ell)}(z) - 1 + \alpha \sum_{S_1,..,S_\alpha \in \mathcal{S}^{(\ell-1)}} \text{node}(S_1)\, z^{\sum_{i=1}^{\alpha} |S_i|}$$

$$= \mathcal{P}^{(\ell)}(z) - 1 + \alpha\, \text{NODE}^{(\ell-1)}(z)\, (\mathcal{P}^{(\ell-1)}(z))^{\alpha-1}$$

Then

$$\frac{\text{NODE}^{(\ell)}(z)}{\mathcal{P}^{(\ell)}(z)} = \alpha\ \text{NODE}^{(\ell-1)}(z)\ \frac{(\mathcal{P}^{(\ell-1)}(z))^{\alpha-1}}{\mathcal{P}^{(\ell)}(z)} + 1 - \frac{1}{\mathcal{P}^{(\ell)}(z)}$$

$$= \alpha\ \frac{\text{NODE}^{(\ell-1)}(z)}{\mathcal{P}^{(\ell-1)}(z)} + 1 - \frac{1}{\mathcal{P}^{(\ell)}(z)}$$

Recursively we obtain

$$\frac{\text{NODE}^{(\ell)}(z)}{\mathcal{P}^{(\ell)}(z)} = \alpha^{\ell}\, \frac{\text{NODE}^{(o)}(z)}{\mathcal{P}^{(o)}(z)} + \sum_{i=0}^{\ell-1} \alpha^{i} \left(1 - \frac{1}{\mathcal{P}^{(\ell-i)}(z)} \right)$$

But $NODE^{(o)}(z) = z$ and $\mathcal{P}^{(o)}(z) = 1 + z$

($\mathcal{S}^{(o)} = \{\emptyset, \{\varepsilon\}\}$ where ε denotes the empty strings)

Then

$$NODE^{(\ell)}(z) = \alpha^{\ell} \frac{z}{1+z} \mathcal{P}^{(\ell)}(z) + \mathcal{P}^{(\ell)}(z) \sum_{i=0}^{\ell-1} \alpha^{i} \left(1 - \frac{1}{\mathcal{P}^{(\ell i)}(z)}\right)$$

Using (1) we obtain

$$NODE^{(\ell)}(z) = \alpha^{\ell} z (1+z)^{\alpha^{\ell}-1} + \frac{\alpha^{\ell}-1}{\alpha-1} (1+z)^{\alpha^{\ell}} - \sum_{i=0}^{\ell-1} \alpha^{i} (1+z)^{\alpha^{\ell}-\alpha^{\ell-i}}$$

$\alpha_n^{(\ell)}$ can be computed as follows :

$$\alpha_n^{(\ell)} = \alpha^{\ell} \binom{\alpha^{\ell}-1}{n-1} + \frac{\alpha^{\ell}-1}{\alpha-1} \binom{\alpha^{\ell}}{n} - \sum_{i=1}^{\ell-1} \alpha^{i} \binom{\alpha^{\ell}-\alpha^{\ell-i}}{n}$$

(if $n > 0$ and $i=0$, the coefficient of z^n in the term $\alpha^{i} (1+z)^{\alpha^{\ell}-\alpha^{\ell-i}}$ is zero and so that the sum in the above formula begins for $i=1$)

Because of the uniformity assumptions, we have :

$$\bar{S}(n,\ell) = \frac{\alpha_n^{(\ell)}}{|\mathcal{S}_n^{(\ell)}|} = \frac{\alpha_n^{(\ell)}}{\binom{\alpha^{\ell}}{n}}$$

$$= n + \frac{\alpha^{\ell}-1}{\alpha-1} - \sum_{i=1}^{\ell-1} \alpha^{i} \frac{\binom{\alpha^{\ell}-\alpha^{\ell-i}}{n}}{\binom{\alpha^{\ell}}{n}}$$

if we assume that α^{ℓ} is much greater than n, we obtain an asymptotic value of $\bar{S}(n,\ell)$:

$$n \ll \alpha^{\ell} \Rightarrow \binom{\alpha^{\ell}}{n} \simeq \frac{\alpha^{\ell n}}{n!}$$

So that

$$\bar{S}(n,\ell) \simeq n + \frac{\alpha^{\ell}-1}{\alpha-1} - \sum_{i=1}^{\ell-1} \alpha^{i} \frac{(\alpha^{\ell}-\alpha^{\ell-i})^{n}}{\alpha^{\ell n}}$$

$$\simeq n + \frac{\alpha^{\ell}-1}{\alpha-1} - \sum_{i=1}^{\ell-1} \alpha^{i} (1 - \frac{1}{\alpha^{i}})^{n}$$

$$\simeq n + \frac{\alpha^{\ell}-1}{\alpha-1} - \sum_{i=1}^{\ell-1} \alpha^{i} \left(\sum_{k=0}^{n} (-1) \binom{n}{k} \frac{1}{\alpha^{ik}} \right)$$

$$\simeq n + \frac{\alpha^{\ell}-1}{\alpha-1} - \sum_{k=0}^{n} (-1)^{k} \binom{n}{k} \sum_{i=1}^{\ell-1} \frac{1}{\alpha^{i(k-1)}}$$

$$\simeq n + \frac{\alpha^{\ell}-1}{\alpha-1} - (\frac{\alpha^{\ell}-1}{\alpha-1} - 1) + n(\ell-1)$$

$$- \sum_{k=2}^{n} (-1)^{k} \binom{n}{k} \left(\frac{1 - \frac{1}{\alpha^{(k-1)\ell}}}{1 - \frac{1}{\alpha^{k-1}}} - 1 \right)$$

Finally, we obtain

$$\bar{S}(n,\ell) \simeq n\ell + 1 - \sum_{k=2}^{n} (-1)^{k} \binom{n}{k} \left(\frac{1}{\alpha^{k-1} - 1} \right)$$

These results can be summarized as follows. Let S denote a subroutine which recognizes n strings belonging to Σ^{ℓ}.

Then :

The minimum size of S is $n+\ell$

The maximum size of S is $n\ell+1$

The average size of S is

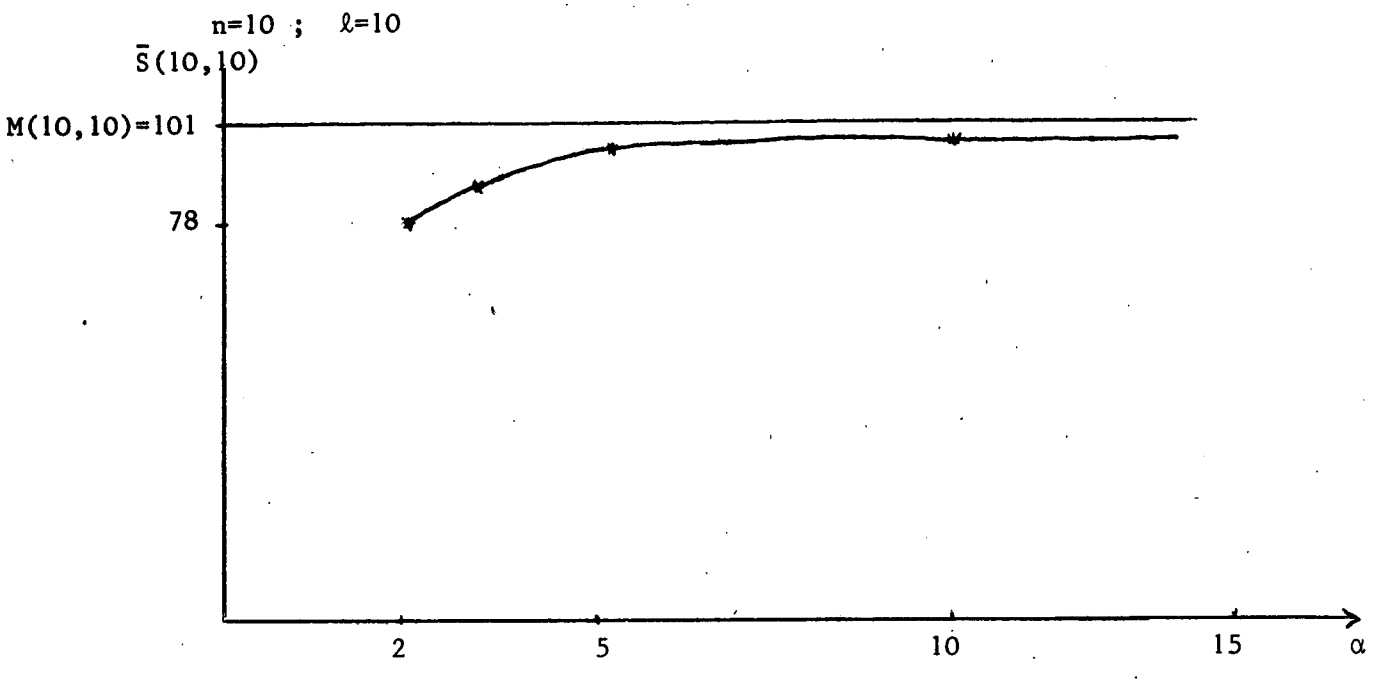
$$n + \frac{\alpha^{\ell}-1}{\alpha-1} + \sum_{i=1}^{\ell-1} \alpha^{i} \frac{\binom{\alpha^{\ell}-\alpha^{\ell-i}}{n}}{\binom{\alpha^{\ell}}{n}}$$

If we also assume $n << \alpha^{\ell}$ then :

The average size of S is

$$n\ell + 1 - \sum_{k=2}^{n} (-1)^{k} \binom{n}{k} \left(\frac{1}{\alpha^{k-1} - 1} \right)$$

These formulae can be visualised as follows :



The most important conclusion about these results, is that the average size of a subroutine is approximatively the maximum size, as soon as the alphabet size is greater than 10.

This means that all the strings are quickly distinct. In particular, we see that the hypothesis of fixed length is not restrictive : the average size of the subroutine which recognizes strings $(\mu_i)_{1 \le i \le n}$ with variable length should be equivalent to the maximum value which is

$$1 + \sum_{i=1}^{n} |\mu_i| \ .$$

Evaluation of the number of logical states

The number of logical states is a function of the condition and of the format which are compiled.

If we want to get an average value of the number of logical states, we have to make assumptions about the kind of condition and the kind of relation format which occur.

But, that condition is a function of the user application : If we just want to query a file, we believe that the condition will have about

two or three elementary conditions. If so, the number of logical states will be small compared to the subroutines size.

But for other applications, we can have very large selection condition :

For instance we might want to compute the semi-join of $R_1(AB)$ and $R_2(AC)$ by projecting R_2 on A and then generating an FSA that recognize all the strings from $R_2(A)$ and filtering R_1 by this FSA.

For such a computation, if $R_2(A)$ is large then so is the selection condition.

So, we shall just propose a maximum value in the general case, and another maximum value for a "saturation" case.

General case :

Let $R(A_1,\ldots,A_K)$ denote a relation and c_i the number of elementary conditions on attribute A_i.

We can notice that each subroutine which filters an attribute A_i has less then (c_i+1) results which are logically different. So, a maximum value of the number of logical case is

$$\prod_{i=1}^{K} (1 + c_i)$$

Saturation case

Let $R(A_1,\ldots,A_N)$ denote a relation and $C = T_1$ or ... or T_k the selection condition.

The saturation means that each T_i, true before a subroutine, can be false after this subroutine, independently from the others. This implies that each T_i have an elementary condition on each attribute.

As said in section I, we have a hierarchical representation of the relational model. So, we can define the depth of an attribute, by the depth of the attribute in the format.

Example : Let R(COURSE,STUDENT,GRADE) denote a relation.

If the format of R is $(COURSE,STUDENT,GRADE)^*$ then all the attribute are in depth 1. If the format of R is $(COURSE(STUDENT(GRADE)^*)^*)^*$ then COURSE is in depth 1, STUDENT in depth 2, and GRADE in depth 3.

Let $d(A_i)$ denote the depth of attribute A_i in relation R ; p the maximum depth of the R attributes.

Let a_i denote the number of R attribute in depth i.

$m(a_1,\ldots,a_p)$ the maximum number of logical states and N_j^i the number of logical states meaning that j terms are true after one of the subroutines which filter an attribute in depth i.

If p=1 $\quad m(a_1) = a_1 * 2^t$

and $N_j^1 = \binom{t}{j}$

Recurrence hypothesis

$$m(a_1,\ldots,a_{p-1}) = \sum_{i=1}^{p-1} a_i\ (i+1)^t$$

$$N_j^{p-1} = \binom{t}{j}\ (p-1)^{t-j}$$

Then $\quad N_j^p = \sum_{\ell=j}^{t} N_\ell^{p-1}\ \binom{\ell}{j}$

$$= \sum_{\ell=j}^{t} \binom{t}{\ell}\binom{\ell}{j}(p-1)^{t-\ell}$$

$$= \sum_{\ell=j}^{t} \binom{t}{j}\binom{t-j}{t-\ell}(p-1)^{t-\ell}$$

$$= \binom{t}{j} \sum_{\ell=0}^{t-j} \binom{t-j}{t-j-\ell}(p-1)^{t-j-\ell}$$

So $N^p_j = \binom{t}{j} p^{t-j}$.

Then $m(a_1,\ldots,a_p) = m(a_1,\ldots,a_{p-1}) + a_p \sum_{K=0}^{t} \binom{t}{K} p^{t-K}$

$= m(a_1,\ldots,a_p) + a_p(p+1)^t$

$= \sum a_i (i+1)^t$

Note that for p=1 which means flat format we have $m(a_i) = a_1 2^t$. It gives a good idea of the complexity introduced by the compacted format.

CONCLUSION

The VERSO filter is presently under implementation and should be operationnal by the end of 82. Until that date, we are using an emulation of the filter implemented in PASCAL on the MULTICS system. The compiler was written on PASCAL and is operationnal. Compiling time is comparable to standard compilation (i.e. to the time needed to generate standard code to perform selection and projection). The size of the automaton tried with "realistic examples" seems to remain reasonnable (less than 50 states) thus showing the feasibility of FSA filtering.

REFERENCE

[1] Le filtrage de Données dans la Machine Base de Données VERSO
F.BANCILHON, M.SCHOLL
JMBD 80 Sophia Antipolis.

[2] Sur quelques applications du filtrage sequentiel
J.ROHMER,
Thèse d'Etat, Univ. de Grenoble - 1980.

[3] Une nouvelle classe de filtres, multi expressions et multifonctions.
P.FAUDEMAY
Institut de Programmation - Projet SIRIUS-SABRE.

[4] Design of a Backend Processor for a Database Machine
F.BANCILHON, M.SCHOLL
(INRIA).

[5] Implementing a Relational Database by Means of Specialized Hardware
E.BABB
ACM Transactions of Database Systems
Vol 4 N° 1 - March 79 - Pages 1-29.

[6] Systeme associatif de Recherche d'informations
A.LECALVE, P.KALFAN
(SINTRA).

[7] Un automate à Pile pour la Recherche d'informations
D.TUSERA
(INRIA) - projet TREFLE.

[8] Operational characteristics of a hardaware-based pattern matcher
HASKIN et HOLLAAR
IBM Research Report, RJ 3229.

[9] The Sure filter
STIEGE et LEILICH
Internal Report.

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique